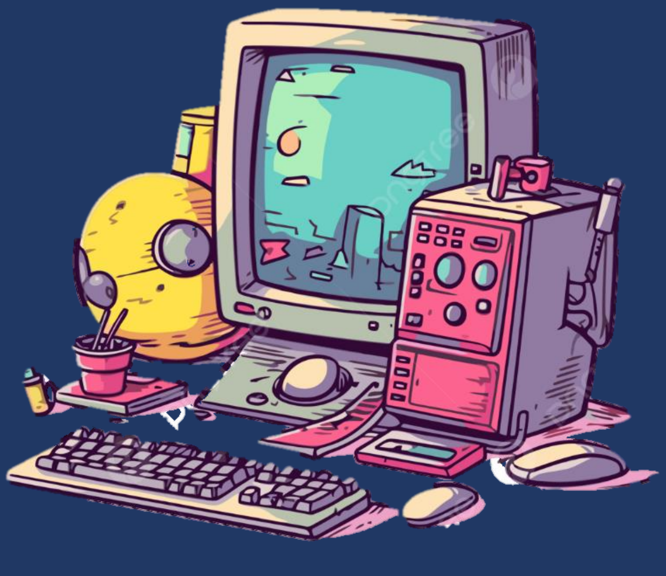
A Summary and Extra Notes On Chapter 2



Order of topics that are going to be covered in this file:

Why using namespace std is considered bad practice?

Back when I was doing a little C++ programming, I remember going to these forums asking questions, I was always hit with this statement:

" I strongly suggest you never use `using namespace std;` and always type out the `std::`, it's more explicit and prevents errors. "

Apparently, this is something considered to cause performance issues. I did some research about this and I found one very good reason why it's considered bad practice from this reddit thread:

"

If you write `using namespace std;` it imports all the symbols inside std, which is considered bad practice especially in header files, because it imports so many symbols that it increases the risks of collisions with your own symbols, or with the symbols of any compilation unit that includes the header.

"

So as you practice, it's best to avoid it. As an example, instead of writing like this:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World\n";
    return 0;
}
```

Write it like this:

```
#include <iostream>

int main() {
    std::cout << "Hello World!\n";
    return 0;
}
```

I leave the resources that I found why it's considered bad practice here. I recommend taking a look at them as some of the answers presented are interesting as well.

https://www.reddit.com/r/cpp_questions/comments/o9gmjc/why_is_using_namespace_std_used_in_c_programs/

https://stackoverflow.com/questions/1452721/whats-the-problem-with-using-namespace-std

https://www.geeksforgeeks.org/using-namespace-std-considered-bad-practice/

What is std_lib_facilities.h mentioned in the book?

The std_lib_facilities.h is a header file that's being used throughout the course. This is something that I dislike being presented in this manner (mainly because it confuses new readers and also every time when you want to write new programs, you have to copy and paste that into your projects folder). This file according to the author will be reviewed in the upcoming chapters. If you follow the drills of chapter 2, This is what's stated about this file:
"
How do you find `std_lib_facilities.h`? If you are in a course, ask your instructor. If not, download it from our support site https://www.stroustrup.com/Programming. But what if you don't have an instructor and no access to the web? In that case (only), replace the `#include` directive with

```
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;
inline void keep_window_open() { char ch; cin>>ch; }
```

"

In reality, you don't need to always be writing these lines mentioned above, at the beginning of each program (except for #include <iostream>, that one is always mandatory). In short, you don't have to include all of these to be precise. I'm gonna explain in a short summarized manner what each of those #include directives are for:

`#include<iostream>`
Responsible for input output stream. The i in the beginning stands for input and o stands for output. This is required to be included for using the functionality of `std::cout` and `std::cin`.

`#include<string>`
As the name suggests, this is responsible for string functionality. However, when declaring a string variable, including this line in your program is not necessary.

`#include<vector>`
This is responsible for vector functionality. Vectors will be explained in chapter 4. When declaring a vector of any type, you must include that.

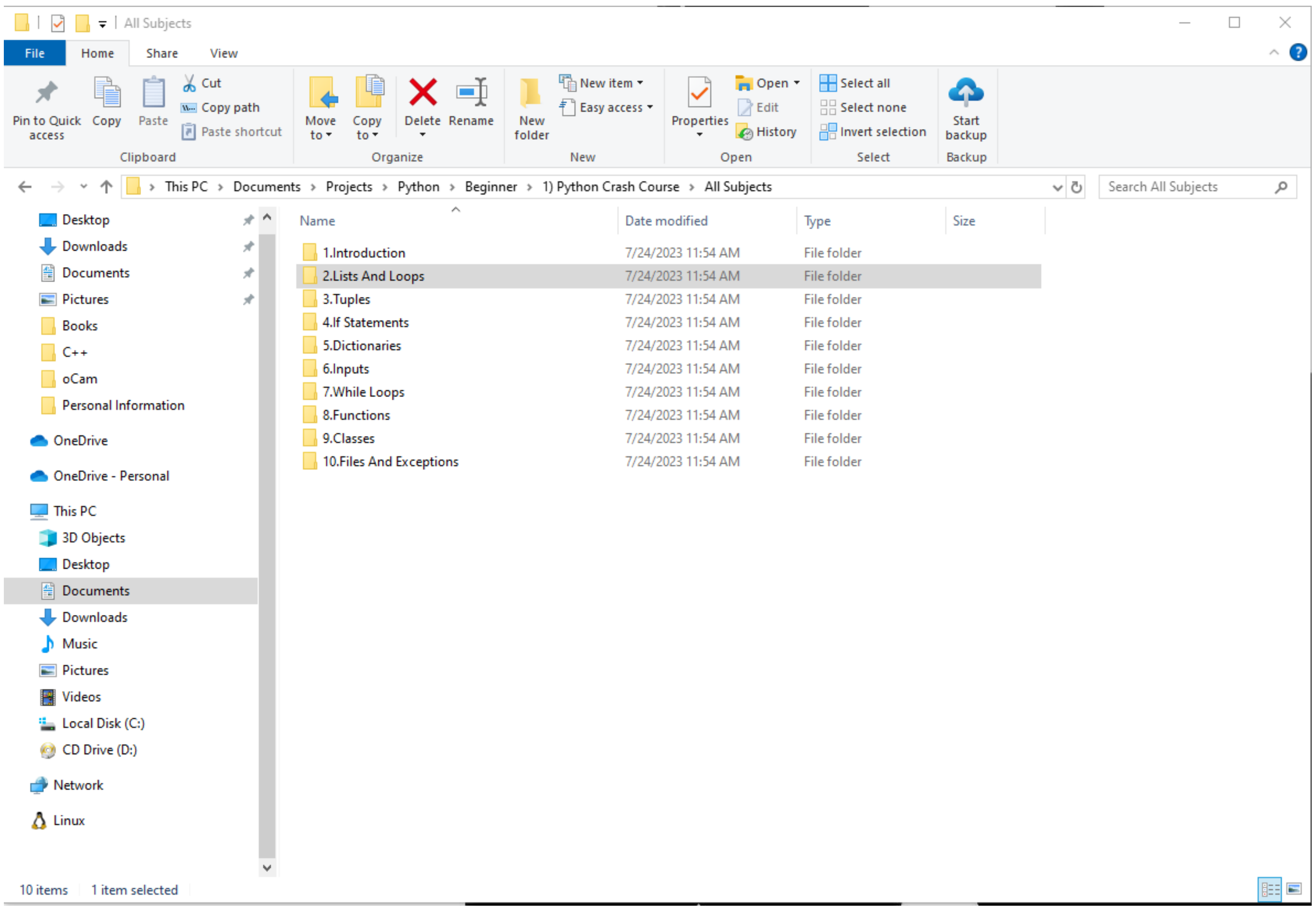
`#include<algorithm>`
This one to be honest, I don't know. Please search it for yourself on the web. LOL :))

`#include<cmath>`
This comes in handy when you want to use the square root functionality.

The one that you're going to use in most cases in the iostream header file.

How to memorize code?

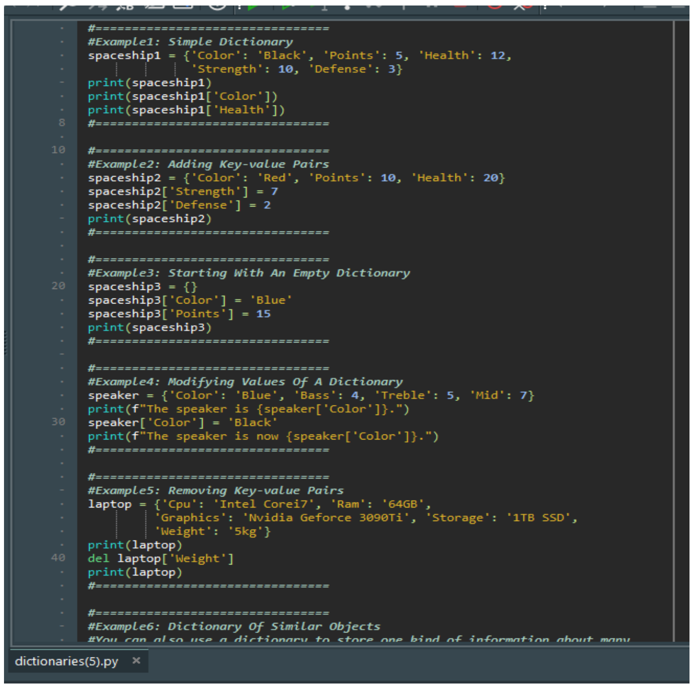
Short answer, you don't. The question to be precise, is that what to do not to forget the code we're going to learn? The answer to this question, lies on the approach you're going to go with. Programming requires practice. So much practice. The best way to not forget code is to create cheat sheets. As you are learning, organize the topics and store code into separate folders. So when you forget something, instead of searching the web, you go back to those files and refresh your memory. I show you an example here. The



Name	Date modified	Type	Size
1.Introduction	7/24/2023 11:54 AM	File folder	
2.Lists And Loops	7/24/2023 11:54 AM	File folder	
3.Tuples	7/24/2023 11:54 AM	File folder	
4.If Statements	7/24/2023 11:54 AM	File folder	
5.Dictionaries	7/24/2023 11:54 AM	File folder	
6.Inputs	7/24/2023 11:54 AM	File folder	
7.While Loops	7/24/2023 11:54 AM	File folder	
8.Functions	7/24/2023 11:54 AM	File folder	
9.Classes	7/24/2023 11:54 AM	File folder	
10.Files And Exceptions	7/24/2023 11:54 AM	File folder	

This is my Python code snippets folder. I organized everything as you can see into separate folders. Let's for example see the dictionaries folder. It contains one python file containing all the necessary examples. You can do the same for C++ as well, once the book is finished. I might do that as well and if done, I'll share it with everyone on Discord :)))

By the way, this is PyScripter IDE in case you're wondering. It's a really simple amazing IDE for Python.



```python
#================================
#Example1: Simple Dictionary
spaceship1 = {'Color': 'Black', 'Points': 5, 'Health': 12,
              'Strength': 10, 'Defense': 3}
print(spaceship1)
print(spaceship1['Color'])
print(spaceship1['Health'])
#================================

#================================
#Example2: Adding Key-value Pairs
spaceship2 = {'Color': 'Red', 'Points': 10, 'Health': 20}
spaceship2['Strength'] = 7
spaceship2['Defense'] = 2
print(spaceship2)
#================================

#================================
#Example3: Starting With An Empty Dictionary
spaceship3 = {}
spaceship3['Color'] = 'Blue'
spaceship3['Points'] = 15
print(spaceship3)
#================================

#================================
#Example4: Modifying Values Of A Dictionary
speaker = {'Color': 'Blue', 'Bass': 4, 'Treble': 5, 'Mid': 7}
print(f"The speaker is {speaker['Color']}.")
speaker['Color'] = 'Black'
print(f"The speaker is now {speaker['Color']}.")
#================================

#================================
#Example5: Removing Key-value Pairs
laptop = {'Cpu': 'Intel Corei7', 'Ram': '64GB',
          'Graphics': 'Nvidia Geforce 3090Ti', 'Storage': '1TB SSD',
          'Weight': '5kg'}
print(laptop)
del laptop['Weight']
print(laptop)
#================================

#================================
#Example6: Dictionary Of Similar Objects
#You can also use a dictionary to store one kind of information about many
```

My favorite quotes on preface & chapter 0

“

Why would you want to program? Our civilization runs on software. Without understanding software you are reduced to believing in “magic” and will be locked out of many of the most interesting, profitable, and socially useful technical fields of work.

“

“

Like mathematics, programming — when done well — is a valuable intellectual exercise that sharpens our ability to think.

“

“

Most of the programming concepts that you will learn using C++ can be used directly in other languages, such as C, C#, Fortran, and Java.

“

“

Learning together and discussing problems with friends is not cheating! It is the most efficient — as well as most pleasant — way of making progress. If nothing else, working with friends forces you to articulate your ideas, which is just about the most efficient way of testing your understanding and making sure you remember.

“

This is why I was insisting so much on reading these two chapters.

“

However, despite the index and the cross-references, this is not a book that you can open to any page and start reading with any expectation of success. Each section and each chapter assume understanding of what came before.

“

Also this is why I encourage everyone to ask as many questions as possible.
“

Learning to ask the right (often hard) questions is an essential part of learning to think as a programmer. Asking only the easy and obvious questions would make you feel good, but it wouldn’t help make you a programmer.
“

“

Learning involves repetition. Our ideal is to make every important point at least twice and to reinforce it with exercises.
“

“

Someone will argue, “We must move slowly and carefully; we must walk before we can run!” But have you ever watched a baby learning to walk? Babies really do run by themselves before they learn the finer skills of slow, controlled walking. Similarly, you will dash ahead, occasionally stumbling, to get a feel of programming before slowing down to gain the necessary finer control and understanding. You must run before you can walk!
“

“

It is essential that you don’t get stuck in an attempt to learn “everything” about some language detail or technique. For example, you could memorize all of C++’s built-in types and all the rules for their use. Of course you could, and doing so might make you feel knowledgeable. However, it would not make you a programmer. Skipping details will get you “burned” occasionally for lack of knowledge, but it is the fastest way to gain the perspective needed to write good programs.
“

“

Knowing “why” is an essential part of acquiring programming skills. Conversely, just memorizing lots of poorly understood rules and language facilities is limiting, a source of errors, and a massive waste of time. We consider your time precious and try not to waste it.
“